Sumário

[1) Problema 2](#_Toc535077928)

[2) Proposta 2](#_Toc535077929)

[2.1) Bibliotecas Utilizadas 2](#_Toc535077930)

[2.2) Arquivos Utilizados 2](#_Toc535077931)

[3) Análise exploratória 3](#_Toc535077932)

[4) Pré processamento 4](#_Toc535077933)

[4.1) Atributos Desnecessários 4](#_Toc535077934)

[4.2) Tratamento de dados Faltantes 4](#_Toc535077935)

[4.3) Transformação dos Atributos 5](#_Toc535077936)

[5) Aplicação dos modelos 7](#_Toc535077937)

[5.1) KNN 7](#_Toc535077938)

[5.2) KNN + LR 8](#_Toc535077939)

[5.3) Árvore de Decisão 8](#_Toc535077940)

[5.4) Árvore de Decisão + LR 8](#_Toc535077941)

[6) Resultado dos modelos 9](#_Toc535077942)

[6.1) KNN 9](#_Toc535077943)

[6.2) KNN + LR 9](#_Toc535077944)

[6.3) Árvore de Decisão 10](#_Toc535077945)

[6.4) Árvore de Decisão + LR 10](#_Toc535077946)

# Problema

O problema consiste em utilizar 27 atributos que descrevem o estado de saúde de cavalos e conseguir descobrir se o cavalo vai sobreviver, morrer ou sofrer eutanásia. Esse problema de classificação é apresentado em 2 arquivos, 1 para o treinamento dos modelos criados e outro para o teste dos modelos.

# Proposta

Para o problema, foi proposto utilizar a linguagem de programação python devido a sua facilidade de implementação e por ser amplamente utilizada na criação de modelos peditórios. A IDE escolhida foi *a Jupyter Notebook* devido a sua praticidade para criar e debugar o código.

## 2.1) Bibliotecas Utilizadas

A principal biblioteca utilizada para a solução do problema é a *pandas*, porém as demais bibliotecas utilizadas estão abaixo.

**from** sklearn**.**model\_selection **import** train\_test\_split

**from** sklearn**.**neighbors **import** KNeighborsClassifier

**from** sklearn**.**feature\_selection **import** RFE

**from** sklearn**.**linear\_model **import** LogisticRegression

**from** sklearn **import** tree

**import** sklearn**.**metrics **as** metrics

**import** sklearn**.**preprocessing **as** preprocessing

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** plt

**import** numpy **as** np

pd**.**options**.**display**.**max\_rows **=** 999

## 2.2) Arquivos Utilizados

Os 2 arquivos utilizados (horse.csv e horseTest.csv) foram lidos diretamente via função *read\_csv* da biblioteca *pandas*. Um é utilizado para a criação e treinamento dos modelos e o outro para o teste dos modelos.

#Load in the data with `read\_csv()`

horsesDataSet **=** pd**.**read\_csv**(**'horse.csv'**,** header**=**0**,** delimiter**=**','**)**

horsesDataSetTest **=** pd**.**read\_csv**(**"horseTest.csv"**,** header**=**0**,** delimiter**=**','**)**

#description of dataSet

descriptionHorsesDataSet **=** horsesDataSet**.**describe**(**include**=**'all'**)**

descriptionHorsesDataSetTest **=** horsesDataSetTest**.**describe**(**include**=**'all'**)**

# 3) Análise exploratória

# Pré processamento

Esta etapa do trabalho consiste em preparar os dados para que os modelos possam ser executados. A etapa pode ser dividida em 3 fases:

1. Analisando atributos desnecessários e/ou redundantes;
2. Tratamento de dados faltantes;
3. Transformação dos atributos para uma melhor análise dos modelos.

## 4.1) Atributos Desnecessários

Nessa fase foi avaliado a quantidade de dados faltantes para cada atributo, e foi decidido deletar e não utilizar atributos que possuem mais dados faltantes que um determinado *threshold* estipulado previamente. O *threshold* utilizado foi de 0.5, ou seja, atributos com mais de 50% de dados faltantes foram deletados da base de treino dos modelos.

# iterate through each attribute and define the percentage of missing values

# populate array with zeros with column dimensions of dataset

qtd\_nan **=** **[**0 **for** x **in** range**(**horsesDataSet**.**shape**[**1**])]**

# populate array with zeros with column dimensions of dataset

qtd\_total **=** **[**0 **for** x **in** range**(**horsesDataSet**.**shape**[**1**])]**

i **=** 0

**while** i **<** horsesDataSet**.**shape**[**1**]:**

# get array of boolean describing each line as null or not for i attribute

attributeLinesIsNA **=** pd**.**isna**(**horsesDataSet**.**iloc**[:,** i**])**

# get current attribute label name

currentAttributeLabel **=** list**(**horsesDataSet**)[**i**]**

qtd\_nan**[**i**]** **=** horsesDataSet**.**loc**[**attributeLinesIsNA**,** currentAttributeLabel**].**shape**[**0**]**

qtd\_total**[**i**]** **=** horsesDataSet**.**loc**[:,** currentAttributeLabel**].**shape**[**0**]**

i **=** i**+**1

percentageArray **=** np**.**divide**(**qtd\_nan**,** qtd\_total**)**

# dropping atributes

threshold **=** 0.5

PreProcessedHorseDataSet **=** horsesDataSet

PreProcessedHorseDataSetTest **=** horsesDataSetTest

i **=** 0

**while** i **<** horsesDataSet**.**shape**[**1**]:**

**if** percentageArray**[**i**]** **>** threshold**:**

# get current attribute label name

currentAttributeLabel **=** list**(**horsesDataSet**)[**i**]**

# drop attribute column if na values > threshold

PreProcessedHorseDataSet **=** PreProcessedHorseDataSet**.**drop**(**columns**=**currentAttributeLabel**)**

#drop from test

PreProcessedHorseDataSetTest **=** PreProcessedHorseDataSetTest**.**drop**(**columns**=**currentAttributeLabel**)**

i **=** i **+** 1

## 4.2) Tratamento de dados Faltantes

Com os atributos restantes, nessa fase foram feitas as inserções de dados para auxiliar os modelos em seus treinamentos. Para atributos numéricos, foi decidido substituir os valores nulos pela média de todos os outros existentes. Para os atributos categóricos, foi decidido substituir os valores faltantes pelo valor mais presente dentre os não nulos. Lembrando que também foi necessário fazer o mesmo método na base de teste, porém os valores utilizados para preencher os campos nulos são provenientes da base de treino.

# fill remaining lines with mean values (only numerical)

PreProcessedHorseDataSet **=** PreProcessedHorseDataSet**.**fillna**(**horsesDataSet**.**mean**())**

# Show Statistics of DataSet

StatisticsPreProcessedHorseDataSet **=** PreProcessedHorseDataSet**.**describe**(**include**=**'all'**)**

# Altering Categorical missing values to Mode Value (value that appear the most often)

i **=** 0

**while** i **<** PreProcessedHorseDataSet**.**shape**[**1**]:**

# return the most frequent value (first index because mode() returns a DataFrame)

attributeMode **=** PreProcessedHorseDataSet**.**mode**().**iloc**[**0**,** i**]**

currentAttributeLabel **=** list**(**PreProcessedHorseDataSet**)[**i**]**

PreProcessedHorseDataSet**[**currentAttributeLabel**]** **=** PreProcessedHorseDataSet**[**currentAttributeLabel**].**fillna**(**attributeMode**)**

i **=** i**+**1

# Altering missing values [DATASET TEST]

#Saving values from train to insret in TEST with variable v

v **=** **[**0 **for** x **in** range**(**horsesDataSet**.**shape**[**1**])]**

i**=**0

**while** i **<** PreProcessedHorseDataSet**.**shape**[**1**]:**

**if** PreProcessedHorseDataSet**.**dtypes**[**i**]** **==** 'O'**:**

v**[**i**]** **=** PreProcessedHorseDataSet**.**mode**().**iloc**[**0**,** i**]**

**else:**

v**[**i**]** **=** PreProcessedHorseDataSet**.**iloc**[**0**,** i**].**mean**()**

currentAttributeLabel **=** list**(**PreProcessedHorseDataSetTest**)[**i**]**

PreProcessedHorseDataSetTest**[**currentAttributeLabel**]** **=** PreProcessedHorseDataSetTest**[**currentAttributeLabel**].**fillna**(**v**[**i**])**

i **=** i**+**1

## 4.3) Transformação dos Atributos

Para finalizar, nesta última fase foram feitas três modificações nas bases de treino e de teste. A primeira modificação foi fazer com que o modelo tivesse que optar por escolher se o cavalo iria viver ou morrer, ou seja, a categoria que classificava que o cavalo seria submetido a eutanásia foi transformada na mesma categoria em que o cavalo morreria.

# Change values from euthanized to died

AttributesHorseDataSet **=** PreProcessedHorseDataSet**.**drop**(**'outcome'**,** axis**=**1**)**

TargetHorseDataSet **=** PreProcessedHorseDataSet**.**loc**[:,** 'outcome'**]**

# mapping 'euthanized' values to 'died' to tune fitting

TargetHorseDataSet **=** TargetHorseDataSet**.**map**(lambda** x**:** 'died' **if** x **==** 'euthanized' **else** x**)**

PreProcessedHorseDataSet **=** pd**.**concat**([**AttributesHorseDataSet**,** TargetHorseDataSet**],** axis**=**1**)**

# Change values from euthanized to died [DATASET TEST]

AttributesHorseDataSetTest **=** PreProcessedHorseDataSetTest**.**drop**(**'outcome'**,** axis**=**1**)**

TargetHorseDataSetTest **=** PreProcessedHorseDataSetTest**.**loc**[:,** 'outcome'**]**

# mapping 'euthanized' values to 'died' to tune fitting

TargetHorseDataSetTest **=** TargetHorseDataSetTest**.**map**(lambda** x**:** 'died' **if** x **==** 'euthanized' **else** x**)**

PreProcessedHorseDataSetTest **=** pd**.**concat**([**AttributesHorseDataSetTest**,** TargetHorseDataSetTest**],** axis**=**1**)**

A Segunda modificação na base foi feita para de fato transformar os atributos categóricos. O método utilizado foi o de binarização, que consiste em criar uma coluna na base para cada valor que o atributo pode possuir preenchendo essas novas colunas com 0 ou 1, onde 1 informa que aquela coluna é o valor do atributo e 0 informa o contrário. Dessa forma, os dados podem passar por qualquer modelo com mais facilidade, uma vez que cada coluna nova só poderá assumir 2 valores.

# categorical attribute binarization

categoricalHorseDataSet **=** PreProcessedHorseDataSet**.**select\_dtypes**(**include**=**'object'**)**

categoricalHorseDataSet **=** categoricalHorseDataSet**.**drop**(**'outcome'**,** axis**=**1**)**

categoricalHorseDataSetDummy **=** pd**.**get\_dummies**(**categoricalHorseDataSet**)**

PreProcessedHorseDataSet **=** pd**.**concat**([**categoricalHorseDataSetDummy**,** PreProcessedHorseDataSet**.**loc**[:,** 'outcome'**]],** axis**=**1**)**

# categorical attribute binarization [DATASET TEST]

categoricalHorseDataSetTest **=** PreProcessedHorseDataSetTest**.**select\_dtypes**(**include**=**'object'**)**

categoricalHorseDataSetTest **=** categoricalHorseDataSetTest**.**drop**(**'outcome'**,** axis**=**1**)**

categoricalHorseDataSetDummy **=** pd**.**get\_dummies**(**categoricalHorseDataSetTest**)**

PreProcessedHorseDataSetTest **=** pd**.**concat**([**categoricalHorseDataSetDummy**,** PreProcessedHorseDataSetTest**.**loc**[:,** 'outcome'**]],** axis**=**1**)**

Quanto a terceira transformação, ela foi realizada para trocar o tipo da variável de saída de *object* para *category*, sabendo que alguns modelos só aceitam esse tipo de variável.

# Convertendo objetos para categóricos

i**=** 0

**while** i **<** PreProcessedHorseDataSet**.**shape**[**1**]:**

**if** PreProcessedHorseDataSet**[**list**(**PreProcessedHorseDataSet**)[**i**]].**dtypes **==** 'O'**:**

PreProcessedHorseDataSet**[**list**(**PreProcessedHorseDataSet**)[**i**]]** **=** PreProcessedHorseDataSet**[**list**(**PreProcessedHorseDataSet**)[**i**]].**astype**(**'category'**)**

i **=** i**+**1

i**=** 0

**while** i **<** PreProcessedHorseDataSetTest**.**shape**[**1**]:**

**if** PreProcessedHorseDataSetTest**[**list**(**PreProcessedHorseDataSetTest**)[**i**]].**dtypes **==** 'O'**:**

PreProcessedHorseDataSetTest**[**list**(**PreProcessedHorseDataSetTest**)[**i**]]** **=** PreProcessedHorseDataSetTest**[**list**(**PreProcessedHorseDataSetTest**)[**i**]].**astype**(**'category'**)**

i **=** i**+**1

# 5) Aplicação dos modelos

Esta etapa do trabalho consiste na criação e aplicação dos modelos de *Machine Learning*. Foram utilizados basicamente 2 modelos, KNN (*k-nearest neighbors*) e Árvore de Decisão, e para cada um deles foi utilizado o método de Regressão Logística (LR) para potencializá-los.

## 5.1) KNN

Esse modelo consiste em associar a saída do objeto testado ao seu vizinho lógico mais próximo, ou seja, procura na base de treino o objeto mais parecido com o testado e desse modo o associa a ele. Como na biblioteca utilizada era obrigatório informar o número vizinhos, foi decidido rodar o mesmo algoritmo variando o número de vizinhos de 0 a 100 e escolhendo o melhor número de vizinhos ótimos àquele que possuísse o melhor resultado.

result\_lr **=** 0

melhor\_qty **=** 0

resultados **=** **[]**

**for** j **in** range**(**1**,**len**(**PreProcessedHorseDataSet**)):**

model **=** LogisticRegression**()**

feature\_qty **=** j

dataset **=** PreProcessedHorseDataSet

rfe **=** RFE**(**model**,** feature\_qty**)**

rfe **=** rfe**.**fit**(**dataset**.**drop**(**'outcome'**,** axis **=** 1**),** dataset**.**outcome**.**values**)**

col **=** **[]**

**for** i **in** range**(**0**,**rfe**.**support\_**.**size**):**

**if** rfe**.**support\_**[**i**]** **==** **True:**

col**.**append**(**i**)**

dataset\_after\_rfe **=** dataset**[**dataset**.**columns**[**col**]]**

x\_train**,** x\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**dataset\_after\_rfe**,** dataset**[**'outcome'**],** random\_state **=** 0**)**

tamanho **=** 100

maior\_knn **=** 0

pos **=** 0

vet **=[]**

**for** i **in** range**(**1**,**tamanho**):**

knn **=** KNeighborsClassifier**(**n\_neighbors **=** i**)**

knn**.**fit**(**x\_train**,** y\_train**)**

result **=** knn**.**score**(**x\_test**,** y\_test**)**

vet**.**append**(**result**)**

**if** result **>** maior\_knn**:**

maior\_knn **=** result

pos **=** i

knn **=** KNeighborsClassifier**(**n\_neighbors **=** pos**)**

knn**.**fit**(**x\_train**,** y\_train**)**

TESTE **=** PreProcessedHorseDataSetTest**[**list**(**dataset**[**dataset**.**columns**[**col**]].**columns**)]**

h **=** knn**.**score**(**TESTE**,** PreProcessedHorseDataSetTest**.**outcome**)**

resultados**.**append**(**h**)**

**if** h **>** result\_lr**:**

result\_lr **=** h

melhor\_qty **=** j

TargetHorseDataSet\_prediction **=** knn**.**predict**(**x\_test**)**

TargetHorseDataSet\_test **=** y\_test

## 5.2) KNN + LR

A única diferença deste modelo para o anterior é o acréscimo do LR. Tal qual foi utilizado para achar a quantidade ótima de atributos para que o modelo funcione da melhor forma. Uma vez que o LR realiza uma classificação dos atributos avaliando quais são os mais relevantes para a solução do problema. Como na biblioteca utilizada era obrigatório informar o número de atributos desejados, foi decidido rodar o mesmo algoritmo variando o número de atributos de 1 até o número total de atributos.

Dessa forma, foi aplicado a variação do KNN de 0 a 100 para cada quantidade de atributos e a quantidade de atributos ótima foi escolhida levando em consideração o melhor score final.

## 5.3) Árvore de Decisão

Esse modelo consiste em classificar os dados criando diversas ramificações baseadas em comparações com algum valor até que ache a “folha” que melhor represente aquele dado.

## 5.4) Árvore de Decisão + LR

Como foi feito com o KNN, foi utilizado o LR para potencializar o valor do modelo anterior. E o score final é referente a quantidade de atributos ótima.

**for** j **in** range**(**1**,**len**(**PreProcessedHorseDataSet**)):**

model **=** LogisticRegression**()**

feature\_qty **=** j

dataset **=** PreProcessedHorseDataSet

rfe **=** RFE**(**model**,** feature\_qty**)**

rfe **=** rfe**.**fit**(**dataset**.**drop**(**'outcome'**,** axis **=** 1**),** dataset**.**outcome**.**values**)**

col **=** **[]**

**for** i **in** range**(**0**,**rfe**.**support\_**.**size**):**

**if** rfe**.**support\_**[**i**]** **==** **True:**

col**.**append**(**i**)**

dataset\_after\_rfe **=** dataset**[**dataset**.**columns**[**col**]]**

# label encoder

labelEncoder **=** preprocessing**.**LabelEncoder**()**

labelEncoder**.**fit**(**TargetHorseDataSet**.**values**)**

TargetHorseEncodedArray **=** labelEncoder**.**transform**(**TargetHorseDataSet**.**values**)**

TargetHorseEncodedDataSet **=** pd**.**DataFrame**(**TargetHorseEncodedArray**,** columns**=[**'outcome'**])**

# split train and test data and target

AttributesHorseDataSet\_train**,** AttributesHorseDataSet\_test**,** TargetHorseDataSet\_train**,** TargetHorseDataSet\_test **=** train\_test\_split**(**dataset\_after\_rfe**,** TargetHorseEncodedDataSet**,** random\_state**=**1**)**

# initialize model parameters

decisionTreeModel **=** tree**.**DecisionTreeClassifier**()**

# fit model using training data

decisionTreeModel**.**fit**(**AttributesHorseDataSet\_train**,** TargetHorseDataSet\_train**)**

# predict our test data using fitted model

TargetHorseDataSet\_prediction **=** decisionTreeModel**.**predict**(**AttributesHorseDataSet\_test**)**

accuracyScore **=** metrics**.**accuracy\_score**(**TargetHorseDataSet\_test**,** TargetHorseDataSet\_prediction**)**

**if** accuracyScore **>** result\_lr**:**

result\_lr **=** accuracyScore

melhor\_qty **=** j

TargetHorseDataSet\_prediction\_Store **=** decisionTreeModel**.**predict**(**AttributesHorseDataSet\_test**)**

TargetHorseDataSet\_test\_Store **=** TargetHorseDataSet\_test

# 6) Resultado dos modelos

Para a avaliação dos modelos foram utilizadas quatro métricas:

1. Acurácia;
2. Recall;
3. Cohen’s Kappa;
4. Matriz de Confusão.

accuracyScore **=** metrics**.**accuracy\_score**(**TargetHorseDataSet\_test\_Store**,** TargetHorseDataSet\_prediction\_Store**)**

**print(**accuracyScore**)**

recallScore **=** metrics**.**recall\_score**(**TargetHorseDataSet\_test\_Store**,** TargetHorseDataSet\_prediction\_Store**,** average**=None)**

**print(**recallScore**)**

kappaScore **=** metrics**.**cohen\_kappa\_score**(**TargetHorseDataSet\_test\_Store**,** TargetHorseDataSet\_prediction\_Store**)**

**print(**kappaScore**)**

#TargetHorseDataSet\_test = labelEncoder.inverse\_transform(TargetHorseDataSet\_test)

#TargetHorseDataSet\_prediction = labelEncoder.inverse\_transform(TargetHorseDataSet\_prediction)

confusionMatrix **=** pd**.**DataFrame**(**

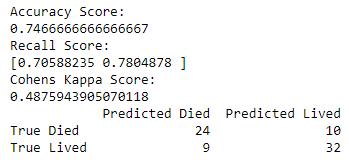
metrics**.**confusion\_matrix**(**TargetHorseDataSet\_test\_Store**,** TargetHorseDataSet\_prediction\_Store**),**

columns**=[**'Predicted Died'**,** 'Predicted Lived'**],**

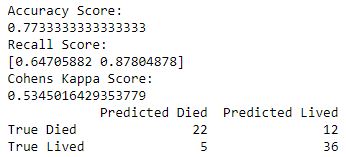
index**=[**'True Died'**,** 'True Lived'**]**

**)**

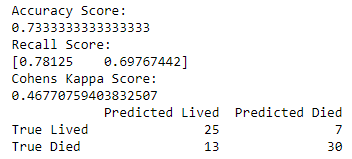
## 6.1) KNN



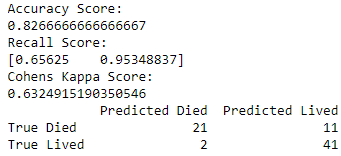
## 6.2) KNN + LR

****

## 6.3) Árvore de Decisão

****

## 6.4) Árvore de Decisão + LR

****